The Differential Analysis of S-functions[⋆,⋆⋆]

Nicky Mouha[⋆⋆⋆], Vesselin Velichkov[†], Christophe De Cannière[‡], and Bart Preneel

[1] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
{Nicky.Mouha,Vesselin.Velichkov,Christophe.DeCanniere}@esat.kuleuven.be

Abstract. An increasing number of cryptographic primitives use operations such as addition modulo 2^n, multiplication by a constant and bitwise Boolean functions as a source of non-linearity. In NIST's SHA-3 competition, this applies to 6 out of the 14 second-round candidates. In this paper, we generalize such constructions by introducing the concept of S-functions. An S-function is a function that calculates the i-th output bit using only the inputs of the i-th bit position and a finite state $S[i]$. Although S-functions have been analyzed before, this paper is the first to present a fully general and efficient framework to determine their differential properties. A precursor of this framework was used in the cryptanalysis of SHA-1. We show how to calculate the probability that given input differences lead to given output differences, as well as how to count the number of output differences with non-zero probability. Our methods are rooted in graph theory, and the calculations can be efficiently performed using matrix multiplications.

Keywords: Differential cryptanalysis, S-function, xdp^+, $\mathrm{xdp}^{\times C}$, $\mathrm{adp}^{\oplus}$, counting possible output differences, ARX.

1 Introduction

Since their introduction to cryptography, differential cryptanalysis [7] and linear cryptanalysis [26] have shown to be two of the most important techniques in both the design and cryptanalysis of symmetric-key cryptographic primitives.

Differential cryptanalysis was introduced by Biham and Shamir in [7]. For block ciphers, it is used to analyze how input differences in the plaintext lead to output differences in the ciphertext. If this happens in a non-random way, this can be used to build a distinguisher or even a key-recovery attack.

The analysis of how differences propagate through elementary components of cryptographic designs is therefore essential to differential cryptanalysis. As typical S-boxes are no larger than 8×8, this analysis can be done by building a difference distribution table. Such a difference distribution table lists the number of occurrences of every combination of input and output differences.

The combination of S-box layers and permutation layers with good cryptographic properties, are at the basis of the wide-trail design. The wide-trail design technique

⋆ The framework proposed in this paper is accompanied by a software toolkit, available at http://www.ecrypt.eu.org/tools

⋆⋆ This work was supported in part by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

⋆⋆⋆ This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

† DBOF Doctoral Fellow, K.U.Leuven, Belgium.

‡ Postdoctoral Fellow of the Research Foundation – Flanders (FWO).

Table 1. Notation.

Notation	Description
$x \parallel y$	concatenation of the strings x and y
$\lvert A \rvert$	number of elements of set A
$x \ll s$	shift of x to the left by s positions
$x \gg s$	shift of x to the right by s positions
$x \lll s$	rotation of x to the left by s positions
$x \ggg s$	rotation of x to the right by s positions
$x + y$	addition of x and y modulo 2^n (in text)
$x \boxplus y$	addition of x and y modulo 2^n (in figures)
$x[i]$	selection: bit (or element) at position i of word x, where $i = 0$ is the least significant bit (element)

is used in AES [10] to provide provable resistance against both linear and differential cryptanalysis attacks.

However, not all cryptographic primitives are based on S-boxes. Another option is to use only operations such as addition modulo 2^n, exclusive or (xor), Boolean functions, bit shifts and bit rotations. For Boolean functions, we assume that the same Boolean function is used for each bit position i of the n-bit input words.

Each of these operations is very well suited for implementation in software, but building a difference distribution table becomes impractical for commonly used primitives where $n = 32$ or $n = 64$. Examples using such constructions include the XTEA block cipher [32], the Salsa20 stream cipher family [5], as well as the hash functions MD5, SHA-1, and 6 out of 14 second-round candidates[3] of NIST's SHA-3 hash function competition [31].

In this paper, we present the first known fully general framework to analyze these constructions efficiently. It is inspired by the cryptanalysis techniques for SHA-1 by De Cannière and Rechberger [12] (clarified in [30]), and by methods introduced by Lipmaa, Wallén and Dumas [23]. The framework is used to calculate the probability that given input differences lead to given output differences, as well as to count the number of output differences with non-zero probability. Our methods are based on graph theory, and the calculations can be efficiently performed using matrix multiplications. We show how the framework can be used to analyze several commonly used constructions.

Notation is defined in Table 1. Section 2 defines the concept of an S-function. This type of function can be analyzed using the framework of this paper. The differential probability xdp^+ of addition modulo 2^n, when differences are expressed using xor, is analyzed in Sect. 3. We show how to calculate xdp^+ with an arbitrary number of inputs. In Sect 4, we study the differential probability $\mathrm{adp}^\oplus$ of xor when differences are expressed using addition modulo 2^n. Counting the number of output differences with non-zero probability is the subject of Sect. 5. We conclude in Sect. 6. The matrices obtained for xdp^+ are listed in Appendix A. We show all possible subgraphs for xdp^+ in Appendix B. In Appendix C, we extend xdp^+ to an arbitrary number of inputs. The computation of $\mathrm{xdp}^{\times C}$ is explained in Appendix D. Appendix E lists the matrices for $\mathrm{adp}^\oplus$.

[3] The hash functions BLAKE [4], Blue Midnight Wish [14], CubeHash [6], Shabal [8], SIMD [20] and Skein [13] can be analyzed using the general framework that is introduced in this paper.

2 S-Functions

In this section, we define S-functions, the type of functions that can be analyzed using our framework. In order to show the broad range of applicability of the proposed technique, we give several examples of functions that follow our definition.

An S-function (short for "state function") accepts n-bit words $a_1, a_2, \ldots, a_k$ and a list of states $S[i]$ (for $0 \leq i < n$) as input, and produces an n-bit output word b in the following way:

$$(b[i], S[i+1]) = f(a_1[i], a_2[i], \ldots, a_k[i], S[i]), \quad 0 \leq i < n \ . \tag{1}$$

Initially, we set $S[0] = 0$. Note that f can be any arbitrary function that can be computed using only input bits $a_1[i], a_2[i], \ldots, a_k[i]$ and state $S[i]$. For conciseness, the same function f is used for every bit $0 \leq i < n$. Our analysis, however, does not require functions f to be the same, and not even to have the same number of inputs. A schematic representation of an S-function is given in Fig. 1.

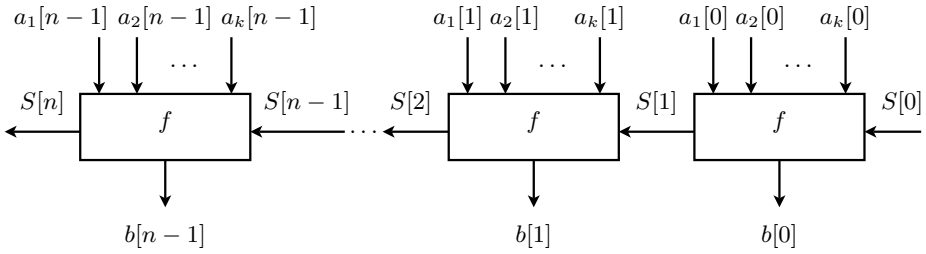

Fig. 1. Representation of an S-function.

Examples of S-functions include addition, subtraction and multiplication by a constant (all modulo 2^n), exclusive-or (xor) and bitwise Boolean functions. Although this paper only analyzes constructions with one output b, the extension to multiple outputs is straightforward. Our technique therefore also applies to larger constructions, such as the Pseudo-Hadamard Transform used in SAFER [1] and Twofish [34], and first analyzed in [21].

With a minor modification, the S-function concept allows the inputs $a_1, a_2, \ldots, a_k$ and the output b to be rotated (or reordered) as well. This corresponds to rotating (or reordering) the bits of the input and output of f. This results in exactly the same S-function, but the input and output variables are relabeled accordingly. An entire step of SHA-1 as well as the `MIX` primitive of the block cipher RC2 can therefore be seen as an S-function. If the extension to multiple output bits is made, this applies as well to an entire step of SHA-2: for every step of SHA-2, two 32-bit registers are updated.

Every S-function is also a *T-function*, but the reverse is not always true. Proposed by Klimov and Shamir [19], a T-function is a mapping in which the i-th bit of the output depends only on bits $0, 1, \ldots, i$ of the input. Unlike a T-function, the definition of an S-function requires that the dependence on bits $0, 1, \ldots, i-1$ of the input can be described by a finite number of states. Therefore, squaring modulo 2^n is a T-function, but not an S-function.

In [11], Daum introduced the concept of a *narrow T-function*. A w-narrow T-function computes the i-th output bit based on some information of length w bits computed from all previous input bits. An S-function, however, requires only the i-th input bit and a state $S[i]$ to calculate the i-th output bit and the next state $S[i+1]$. There is a subtle difference between narrow T-functions and S-functions. If the number

of states is finite and not dependent on the word length n, it may not always be possible for a narrow T-function to compute $S[i+1]$ from the previous state $S[i]$ and the i-th input bit.

It is possible to simulate every S-function using a *finite-state machine* (FSM), also known as a finite-state automaton (FSA). This finite-state machine has k inputs $a_1[i], a_2[i], \ldots, a_k[i]$, and one state for every value of $S[i]$. The output is $b[i]$. The FSM is clocked n times, for $0 \leq i < n$. From (1), we see that the output depends on both the current state and the input. The type of FSM we use is therefore a Mealy machine [27].

The straightforward hardware implementation of an S-function corresponds to a *bit-serial* design. Introduced by Lyon in [24,25], a bit-serial hardware architecture treats all n bits in sequence on a single hardware unit. Every bit requires one clock cycle to be processed.

The S-function framework can also be used in differential cryptanalysis, when the inputs and outputs are xor- or additive differences. Assume that every input pair (x_1, x_2) satisfies a difference $\Delta^{\bullet}x$, using some group operator $\bullet$. Then, if both x_1 and $\Delta^{\bullet}x$ are given, we can calculate $x_2 = x_1 \bullet \Delta^{\bullet}x$. It is then straightforward to define a function to calculate the output values and the output difference as well. This approach will become clear in the following sections, when we calculate the differential probabilities xdp^+ and $\text{adp}^{\oplus}$ of modular addition and xor respectively.

3 Computation of xdp^+

3.1 Introduction

In this section, we study the differential probability xdp^+ of addition modulo 2^n, when differences are expressed using xor. Until [22], no algorithm was published to compute xdp^+ faster than exhaustive search over all inputs. In [22], the first algorithm with a linear time in the word length n was proposed. If n-bit computations can be performed, the time complexity of this algorithm becomes sublinear in n.

In [23], xdp^+ is expressed using the mathematical concept of rational series. It is shown that this technique is more general, and can also be used to calculate the differential probability $\text{adp}^{\oplus}$ of xor, when differences are expressed using addition modulo 2^n.

In this paper, we present a new technique for the computation of xdp^+, using graph theory. The main advantage of the proposed method over existing techniques, is that it is not only more general, but also allows results to be obtained in a fully automated way. The only requirement is that both the operations and the input and output differences of the cryptographic component can be written as the S-function of Sect. 2. In the next section, we introduce this technique to calculate the probability xdp^+.

3.2 Defining the Probability xdp^+

Given n-bit words $x_1, y_1, \Delta^{\oplus}x, \Delta^{\oplus}y$, we calculate $\Delta^{\oplus}z$ using

$$x_2 \leftarrow x_1 \oplus \Delta^{\oplus}x \ , \tag{2}$$

$$y_2 \leftarrow y_1 \oplus \Delta^{\oplus}y \ , \tag{3}$$

$$z_1 \leftarrow x_1 + y_1 \ , \tag{4}$$

$$z_2 \leftarrow x_2 + y_2 \ , \tag{5}$$

$$\Delta^{\oplus}z \leftarrow z_2 \oplus z_1 \ . \tag{6}$$

We then define $\mathrm{xdp}^+(\alpha, \beta \to \gamma)$ as

$$\mathrm{xdp}^+(\alpha, \beta \to \gamma) = \frac{|\{(x_1, y_1) : \Delta^{\oplus} x = \alpha, \Delta^{\oplus} y = \beta, \Delta^{\oplus} z = \gamma\}|}{|\{(x_1, y_1) : \Delta^{\oplus} x = \alpha, \Delta^{\oplus} y = \beta\}|} \ , \tag{7}$$

$$= 4^{-n} |\{(x_1, y_1) : \Delta^{\oplus} x = \alpha, \Delta^{\oplus} y = \beta, \Delta^{\oplus} z = \gamma\}| \ , \tag{8}$$

as there are $2^n \cdot 2^n = 4^n$ combinations for the two n-bit words (x_1, y_1).

3.3 Constructing the S-Function for xdp$^+$

We rewrite (2)-(6) on a bit level, using the formulas for multiple-precision addition in radix 2 [28, §14.2.2]:

$$x_2[i] \leftarrow x_1[i] \oplus \Delta^{\oplus} x[i] \ , \tag{9}$$

$$y_2[i] \leftarrow y_1[i] \oplus \Delta^{\oplus} y[i] \ , \tag{10}$$

$$z_1[i] \leftarrow x_1[i] \oplus y_1[i] \oplus c_1[i] \ , \tag{11}$$

$$c_1[i+1] \leftarrow (x_1[i] + y_1[i] + c_1[i]) \gg 1 \ , \tag{12}$$

$$z_2[i] \leftarrow x_2[i] \oplus y_2[i] \oplus c_2[i] \ , \tag{13}$$

$$c_2[i+1] \leftarrow (x_2[i] + y_2[i] + c_2[i]) \gg 1 \ , \tag{14}$$

$$\Delta^{\oplus} z[i] \leftarrow z_2[i] \oplus z_1[i] \ , \tag{15}$$

where carries $c_1[0] = c_2[0] = 0$. Let us define

$$S[i] \leftarrow (c_1[i], c_2[i]) \ , \tag{16}$$

$$S[i+1] \leftarrow (c_1[i+1], c_2[i+1]) \ . \tag{17}$$

Then, (9)-(15) correspond to the S-function

$$(\Delta^{\oplus} z[i], S[i+1]) = f(x_1[i], y_1[i], \Delta^{\oplus} x[i], \Delta^{\oplus} y[i], S[i]), \quad 0 \le i < n \ . \tag{18}$$

Because we are adding two words in binary, both carries $c_1[i]$ and $c_2[i]$ can be either 0 or 1.

3.4 Computing the Probability xdp$^+$

In this section, we use the S-function (18), defined by (9)-(15), to compute xdp^+. We explain how this probability can be derived from the number of paths in a graph, and then show how to calculate xdp^+ using matrix multiplications.

Graph Representation. For $0 \le i \le n$, we will represent every state $S[i]$ as a vertex in a graph (Fig. 2). This graph consists of several subgraphs, containing only vertices $S[i]$ and $S[i+1]$ for some bit position i. We repeat the following for all combinations of $(\alpha[i], \beta[i], \gamma[i])$:

Set $\alpha[i] \leftarrow \Delta^{\oplus} x[i]$ and $\beta[i] \leftarrow \Delta^{\oplus} y[i]$. Then, loop over all values of $(x_1[i], y_1[i], S[i])$. For each combination, $\Delta^{\oplus} z[i]$ and $S[i]$ are uniquely determined by (18). We draw an edge between $S[i]$ and $S[i+1]$ in the subgraph, if and only if $\Delta^{\oplus} z[i] = \gamma[i]$. Note that several edges may have the same set of endpoints.

For completeness, all subgraphs for xdp^+ are given in Appendix B. Let α, β, γ be given. As shown in Fig. 2, we construct a full graph containing all vertices $S[i]$ for $0 \le i \le n$, where the edges between these vertices correspond to those of the subgraphs for $\alpha[i], \beta[i], \gamma[i]$.

Theorem 1. *Let P be the set of all paths from $(c_1[0], c_2[0]) = (0,0)$ to any of the four vertices $(c_1[n], c_2[n]) \in \{(0,0),(0,1),(1,0),(1,1)\}$ (see Fig. 2). Then, there is exactly one path in P for every pair (x_1, y_1) of the set in the definition of* xdp^+*, given by (8).*

Proof. Given $x_1[i]$, $y_1[i]$, $\Delta^{\oplus}x[i]$, $\Delta^{\oplus}y[i]$, $c_1[i]$ and $c_2[i]$, the values of $\Delta^{\oplus}z[i]$, $c_1[i{+}1]$ and $c_2[i+1]$ are uniquely determined by (9)-(15). All paths in P start at $(c_1[0], c_2[0]) = (0,0)$, and only consist of vertices $(c_1[i], c_2[i])$ for $0 \le i \le n$ that satisfy (9)-(15). Furthermore, edges for which $\Delta^{\oplus}z[i] \neq \gamma[i]$ are not in the graph, and therefore not part of any path P. Thus by construction, P contains every pair (x_1, y_1) of the set in (8) exactly once. □

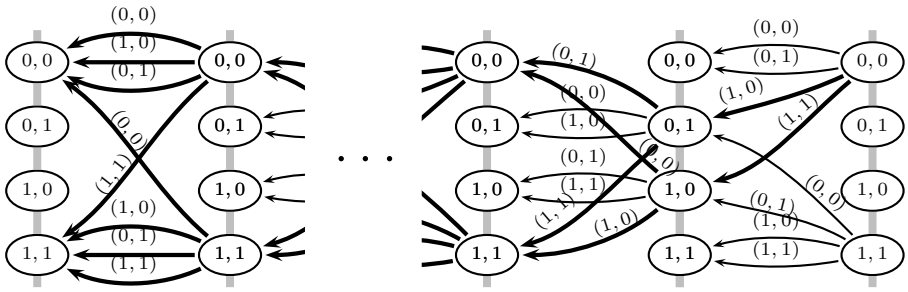

Fig. 2. An example of a full graph for xdp^+. Vertices $(c_1[i], c_2[i]) \in \{(0,0),(0,1),(1,0),(1,1)\}$ correspond to states $S[i]$. There is one edge for every input pair (x_1, y_1). All paths that satisfy input differences α, β and output difference γ are shown in bold. They define the set of paths P of Theorem 1.

Multiplication of Matrices. The differential $(\alpha[i], \beta[i] \to \gamma[i])$ at bit position i is written as a bit string $w[i] \leftarrow \alpha[i] \parallel \beta[i] \parallel \gamma[i]$. Each $w[i]$ corresponds to a subgraph of Appendix B. As this subgraph is a bipartite graph, we can construct its biadjacency matrix $A_{w[i]} = [x_{kj}]$, where x_{kj} is the number of edges that connect vertices $j = S[i]$ and $k = S[i+1]$. These matrices are given in Appendix A.

Let the number of states $S[i]$ be N. Define $1 \times N$ matrix $L = [\,1\ 1 \cdots\ 1\,]$ and $N \times 1$ matrix $C = [\,1\ 0 \cdots\ 0\,]^T$. For any directed acyclic graph, the number of paths between two vertices can be calculated as a matrix multiplication [9]. We can therefore calculate the number of paths P as

$$|P| = L A_{w[n-1]} \cdots A_{w[1]} A_{w[0]} C \ . \tag{19}$$

Using (8), we find that $\mathrm{xdp}^+(\alpha, \beta \to \gamma) = 4^{-n}|P|$. Therefore, we can define $A^*_{w[i]} = A_{w[i]}/4$, and obtain

$$\mathrm{xdp}^+(\alpha, \beta \to \gamma) = L A^*_{w[n-1]} \cdots A^*_{w[1]} A^*_{w[0]} C \ . \tag{20}$$

As such, we obtain a similar expression as in [23], where the xdp^+ was calculated using the concept of rational series. Our matrices $A^*_{w[i]}$ are of size 4×4 instead of 2×2, however. We now give a simple algorithm to reduce the size of our matrices.

3.5 Minimizing the Size of the Matrices for xdp^+.

Corresponding to (20), we can define a non-deterministic finite-state automaton (NFA) with states $S[i]$ and inputs $w[i]$. Compared to a deterministic finite-state automaton,

the transition *function* is replaced by a transition *relation*. There are several choices for the next state, each with a certain probability. This NFA can be minimized as follows.

First, we remove non-accessible states. A state is said to be non-accessible, if it can never be reached from the initial state $S[0] = 0$. This can be done using a simple algorithm to check for connectivity, with a time complexity that is linear in the number of edges.

Secondly, we merge indistinguishable states. The method we propose, is similar to the FSM reduction algorithms found independently by [17] and [29]. Initially, we assign all states $S[i]$ to one equivalence class $T[i] = 0$. We try to partition this equivalence class into smaller classes, by repeating the following steps:

- We iterate over all states $S[i]$.
- For every input $w[i]$ and every equivalence class $T[i]$, we sum the transition probabilities to every state $S[i]$ of this equivalence class.
- If these sums are different for two particular states $S[i]$, we partition them into different equivalence classes $T[i]$.

The algorithm stops when the equivalence classes $T[i]$ cannot be partitioned further.

In the case of xdp^+, we find that all states are accessible. However, there are only two indistinguishable states: $T[i] = 0$ and $T[i] = 1$ when $(c_1[i], c_2[i])$ are elements of the sets $\{(0,0),(1,1)\}$ and $\{(0,1),(1,0)\}$ respectively. Our algorithm shows how matrices $A^*_{w[i]}$ of (20) can be reduced to matrices $A'_{w[i]}$ of size 2×2. These matrices are the same as in [23], but they have now been obtained in an automated way. For completeness, they are given again in Appendix A. Our approach also allows a new interpretation of matrices $A'_{w[i]}$ in the context of S-functions (18): every matrix entry defines the transition probability between two sets of states, where all states of one set were shown to be equivalent by the minimization algorithm.

3.6 Extensions of xdp^+

In this section, we show how S-functions not only lead to expressions to calculate $\mathrm{xdp}^+(\alpha, \beta \rightarrow \gamma)$, but can be applied to related constructions as well.

Multiple Inputs $\mathrm{xdp}^+(\alpha, \beta, \ldots \rightarrow \gamma)$. Using the framework of this paper, we can easily calculate xdp^+ for more than two (independent) inputs. This calculation can be used, for example, in the differential cryptanalysis of XTEA [32] using xor differences. In [15], a 3-round iterative characteristic $(\alpha, 0) \rightarrow (\alpha, 0)$ is used, where $\alpha =$ `0x80402010`. In the third round of the characteristic, there are two consecutive applications of addition modulo 2^n. Separately, these result in probabilities $\mathrm{xdp}^+(\alpha, 0 \rightarrow \alpha) = 2^{-3}$ and $\mathrm{xdp}^+(\alpha, \alpha \rightarrow 0) = 2^{-3}$. It is shown in [15] that the joint probability $\mathrm{xdp}^+(\alpha, 0, \alpha \rightarrow 0)$ is higher than the product of the probabilities $2^{-3} \cdot 2^{-3} = 2^{-6}$, and is estimated to be $2^{-4.755}$. Using the techniques presented in this paper, we evaluate the exact joint probability to be 2^{-3}. We also verified this experimentally. The calculations are detailed in Appendix C. This result can be trivially confirmed using the commutativity property of addition: $\mathrm{xdp}^+(\alpha, \alpha \rightarrow 0) \cdot \mathrm{xdp}^+(0, 0 \rightarrow 0) = \mathrm{xdp}^+(\alpha, \alpha \rightarrow 0) = 2^{-3}$. Nevertheless, our method is more general and can be used for any input difference.

Multiplication by a Constant $\mathrm{xdp}^{\times C}$. A problem related to xdp^+, is the differential probability of multiplication by a constant C where differences are expressed by xor. We denote this probability by $\mathrm{xdp}^{\times C}$. In the hash function Shabal [8], multiplications by 3 and 5 occur. EnRUPT [33] uses a multiplication by 9. In the cryptanalysis of EnRUPT [18], a technique is described to calculate $\mathrm{xdp}^{\times 9}$. This technique is based on a precursor of the framework in this paper. In Appendix D, we show how each of these probabilities can be calculated efficiently, using the framework of this paper. The example of $\mathrm{xdp}^{\times 3}$ is fully worked out.

Pseudo-Hadamard Transform $\mathrm{xdp}^{\mathrm{PHT}}$. The Pseudo-Hadamard Transform (PHT) is defined as $\mathrm{PHT}(x_1, x_2) = (2x_1 + x_2, x_1 + x_2)$. It is a reversible operation, used to provide diffusion in several cryptographic primitives, including block ciphers SAFER [1] and Twofish [34]. Its differential properties were first studied in [21]. If we allow an S-function to be constructed with two outputs b_1 and b_2, the analysis of this construction becomes straightforward using the techniques of this paper.

Step Functions of the MD4 Family. The MD4 family consists of several hash functions, including MD4, MD5, SHA-1, SHA-2 and HAS-160. Currently, the most commonly used hash functions worldwide are MD5 and SHA-1. The step functions of MD4, HAS-160 and SHA-1 can each be represented as an S-function. This applies as well to the `MIX` primitive of the block cipher RC2. They can therefore also be analyzed using our framework. The calculation of the uncontrolled probability $P_u(i)$ in the cryptanalysis of SHA-1 [12,30] uses a precursor of the techniques in this paper. By making the extension to multiple outputs, the same analysis can be made as well for the step function of SHA-2.

4 Computation of $\mathrm{adp}^{\oplus}$

4.1 Introduction

In this section, we study the differential probability $\mathrm{adp}^{\oplus}$ of xor when differences are expressed using addition modulo 2^n. The best known algorithm to compute $\mathrm{adp}^{\oplus}$ was exhaustive search over all inputs, until an algorithm with a linear time in n was proposed in [23].

We show how the technique introduced in Sect. 3 for xdp^{+} can also be applied to $\mathrm{adp}^{\oplus}$. Using this, we confirm the results of [23]. The approach we introduced in this section is conceptually much easier than [23], and can easily be generalized to other constructions with additive differences.

4.2 Defining the Probability $\mathrm{adp}^{\oplus}$

Given n-bit words $x_1, y_1, \Delta^+x, \Delta^+y$, we calculate Δ^+z using

$$x_2 \leftarrow x_1 + \Delta^+x \ , \tag{21}$$
$$y_2 \leftarrow y_1 + \Delta^+y \ , \tag{22}$$
$$z_1 \leftarrow x_1 \oplus y_1 \ , \tag{23}$$
$$z_2 \leftarrow x_2 \oplus y_2 \ , \tag{24}$$
$$\Delta^+z \leftarrow z_2 - z_1 \ . \tag{25}$$

Similar to (8), we define $\mathrm{adp}^{\oplus}(\alpha, \beta \rightarrow \gamma)$ as

$$\mathrm{adp}^{\oplus}(\alpha, \beta \rightarrow \gamma) = \frac{|\{(x_1, y_1) : \Delta^+x = \alpha, \Delta^+y = \beta, \Delta^+z = \gamma\}|}{|\{(x_1, y_1) : \Delta^+x = \alpha, \Delta^+y = \beta\}|} \ , \tag{26}$$
$$= 4^{-n}|\{(x_1, y_1) : \Delta^+x = \alpha, \Delta^+y = \beta, \Delta^+z = \gamma\}| \ , \tag{27}$$

as there are $2^n \cdot 2^n = 4^n$ combinations for the two n-bit words (x_1, y_1).

4.3 Constructing the S-function for $\mathrm{adp}^{\oplus}$

We rewrite (21)-(25) on a bit level, again using the formulas for multiple-precision addition and subtraction in radix 2 [28, §14.2.2]:

$$x_2[i] \leftarrow x_1[i] \oplus \Delta^+ x[i] \oplus c_1[i] \ , \tag{28}$$
$$c_1[i+1] \leftarrow (x_1[i] + \Delta^+ x[i] + c_1[i]) \gg 1 \ , \tag{29}$$
$$y_2[i] \leftarrow y_1[i] \oplus \Delta^+ y[i] \oplus c_2[i] \ , \tag{30}$$
$$c_2[i+1] \leftarrow (y_1[i] + \Delta^+ y[i] + c_2[i]) \gg 1 \ , \tag{31}$$
$$z_1[i] \leftarrow x_1[i] \oplus y_1[i] \ , \tag{32}$$
$$z_2[i] \leftarrow x_2[i] \oplus y_2[i] \ , \tag{33}$$
$$\Delta^+ z[i] \leftarrow (z_2[i] \oplus z_1[i] \oplus c_3[i])[0] \ , \tag{34}$$
$$c_3[i+1] \leftarrow (z_2[i] - z_1[i] + c_3[i]) \gg 1 \ , \tag{35}$$

where carries $c_1[0] = c_2[0] = 0$ and borrow $c_3[0] = 0$. We assume all variables to be integers in two's complement notation, all shifts are signed shifts. Let us define

$$S[i] \leftarrow (c_1[i], c_2[i], c_3[i]) \ , \tag{36}$$
$$S[i+1] \leftarrow (c_1[i+1], c_2[i+1], c_3[i+1]) \ . \tag{37}$$

Then (28)-(35) correspond to the S-function

$$(\Delta^+ z[i], S[i+1]) = f(x_1[i], y_1[i], \Delta^+ x[i], \Delta^+ y[i], S[i]), \qquad 0 \le i < n \ . \tag{38}$$

Both carries $c_1[i]$ and $c_2[i]$ can be either 0 or 1; borrow $c_3[i]$ can be either 0 or -1.

4.4 Computing the Probability $\mathrm{adp}^{\oplus}$

Using the description of the S-function (38), the calculation of $\mathrm{adp}^{\oplus}$ follows directly from Sect. 3.4. We obtain eight matrices $A_{w[i]}$ of size 8×8. After applying the minimization algorithm of Sect. 3.5, the size of the matrices remains unchanged. For completeness, these matrices are given in Appendix E. Here, we use the expression $-4 \cdot c_3[i] + 2 \cdot c_2[i] + c_1[i]$ as an index to order the states $S[i]$. The matrices we obtain are then permutation similar to those of [23]; their states $S'[i]$ can be related to our states $S[i]$ by permutation σ:

$$\sigma = \begin{pmatrix} 0\,1\,2\,3\,4\,5\,6\,7 \\ 0\,4\,2\,6\,1\,5\,3\,7 \end{pmatrix} \ . \tag{39}$$

We calculate the number of paths using (19). From (27), we get $\mathrm{adp}^{\oplus}(\alpha, \beta \to \gamma) = 4^{-n}|P|$. Therefore, we can define $A^*_{w[i]} = A_{w[i]}/4$, and obtain

$$\mathrm{adp}^{\oplus}(\alpha, \beta \to \gamma) = L A^*_{w[n-1]} \cdots A^*_{w[1]} A^*_{w[0]} C \ . \tag{40}$$

5 Counting Possible Output Differences

5.1 Introduction

In the previous sections, we showed for several constructions how to calculate the probability that given input differences lead to a given output difference. A related problem is to calculate the number of *possible* output differences, when the input differences are given. We say that an output difference is possible, if it occurs with a non-zero probability.

First, we describe a naive algorithm to count the number of output differences. It has a time complexity that is exponential in the word length n. We investigate both improvements in existing literature, as well as cryptanalysis results where such a calculation is necessary.

Then, we introduce a new algorithm. We found it to be the first in existing literature with a time complexity that is linear in n. We show that our algorithm can be used for all constructions based on S-functions.

5.2 Algorithm with a Exponential Time in n

Generic Exponential-in-n Time Algorithm. A naive, but straightforward algorithm works as follows. All output differences with non-zero probability can be represented in a search tree. Every level in this tree contains nodes of one particular bit position, with the least significant bit at the top level. This tree is traversed using depth-first search. For each output difference with non-zero probability that is found, we increment a counter for the number of output differences by one. When all nodes are traversed, this counter contains the total number of possible output differences. The time complexity of this algorithm is exponential in n, the memory complexity is linear in n.

Improvement for $\mathbf{xdc^+(\alpha, \beta)}$. We introduce the notation $\mathrm{xdc}^+(\alpha, \beta)$ for the number of output xor-differences of addition modulo 2^n, given input xor-differences α and β. In [3], xdc^+ was used to build a key-recovery attack on top of a boomerang distinguisher for 32-round Threefish-512 [13]. They introduced a new algorithm to calculate xdc^+. The correctness of this algorithm is proven in the full version of [3], i.e. [2]. The algorithm, however, only works if one of the inputs contains either no difference, or a difference only in the most significant bit. Also, it does not generalize to other types of differences. The time complexity of this algorithm is exponential in the number of non-zero input bits, and the memory complexity is linear in the number of non-zero input bits. As a result, it is only usable in practice for sparse input differences. We were unable to find any other work on this problem in existing literature.

5.3 Algorithm with a Linear Time in n

In Sect. 3 and 4, we showed how to calculate the probability of an output difference using both graph theory and matrix multiplications. We now present a similar method to calculate the number of possible output differences. First, the general algorithm is explained. It is applicable to any type of construction based on S-functions. Then, we illustrate how the matrices for xdp^+ can be turned into matrices for xdc^+. This paper is the first to present an algorithm for this problem with a linear-in-n time complexity. We also extend the results to $\mathrm{adp}^{\oplus}$. Our strategy is similar to the calculation of the controlled probability $P_c(i)$, used in the cryptanalysis of SHA-1 [12,30].

Graph Representation. As in Sect. 3.4, we will again construct a graph. Let N be the number of states $|T[i]|$ that we obtained in Sect. 3.5. For xdp^+, we found $N = 2$. We will now construct larger subgraphs, where the nodes do not represent states $T[i]$, but elements of its power set $\mathcal{P}(T[i])$. This power set $\mathcal{P}(T[i])$ contains 2^N elements, ranging from the empty set $\varnothing$ to set of all states $\{0, 1, \ldots, N-1\}$. In automata theory, this technique is known as the subset construction [16, §2.3.5]. It converts the nondeterministic finite-state automaton (NFA) of Sect. 3.5 into a deterministic finite-state automaton (DFA).

For every subgraph, the input difference bits $\alpha[i]$ and $\beta[i]$ are fixed. We then define exactly one edge for every output bit $\gamma[i]$ from every set in $\mathcal{P}(T[i])$ to the corresponding set of next states in $\mathcal{P}(T[i+1])$. The example in the next section will clarify this step.

Theorem 2. *Let P be the set of all paths that start in $\{0\}$ at position $i = 0$ and end in a non-empty set at position $i = n$. Then, the number of paths $|P|$ corresponds to the number of possible output differences.*

Proof. All paths P start in $\{0\}$ at $i = 0$, and end in a non-empty set at $i = n$. For a given output difference bit, there is exactly one edge leaving from a non-empty set of states to another non-empty set of states. Therefore by construction, every possible output difference corresponds to exactly one path in P. □

Multiplication of Matrices. The differential $(\alpha[i], \beta[i])$ at bit position i is written as a bit string $w[i] \leftarrow \alpha[i] \parallel \beta[i]$. As in Sect. 3.4, we construct the biadjacency matrices of these subgraphs. They will be of size $2^N \times 2^N$. As we are only interested in possible output differences, these matrices can be reduced to matrices $B_{w[i]}$ of size $(2^N - 1) \times (2^N - 1)$ by removing the empty set $\varnothing$.

Define $1 \times (2^N - 1)$ matrix $L = [\,1\ 1 \cdots\ 1\,]$ and $(2^N - 1) \times 1$ matrix $C = [\,1\ 0 \cdots\ 0\,]^T$. Similar to (19), we obtain the number of possible output differences as

$$|P| = LB_{w[n-1]} \cdots B_{w[1]} B_{w[0]} C \ . \tag{41}$$

The time complexity of (41) is linear in the word length n.

We note that these matrices can have large dimensions. However, this is often not a problem in practice, as they are typically very sparse. If we keep track of only non-zero elements, there is little memory required to store vectors, and fast algorithms exist for sparse matrix-vector multiplications. Also, the size of the matrices can be minimized using Sect. 3.5.

5.4 Computing the number of output differences xdc$^+$

In the minimized matrices for xdp$^+$ (given in [23] and again in Appendix A), we refer to the states corresponding to the first and the second column as $S[i] = 0$ and $S[i] = 1$ respectively. Then, the subgraphs for xdc$^+$ can be constructed as in Fig. 3. Regardless of the value of the output bit, edges leaving from the empty set $\varnothing$ at i will always arrive at the empty set at $i + 1$. Assume that the input differences are $\alpha[i] = \beta[i] = 0$, and that we are in state $S[i] = 1$, represented in Fig. 3 as $\{1\}$. Recall that the matrices for xdp$^+$ are

$$A'_{000} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \ \ A'_{001} = \frac{1}{2} \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \tag{42}$$

for output differences $\gamma[i] = 0$ and $\gamma[i] = 1$ respectively. To find out which states can be reached from state $S[i] = 1$, we multiply both matrices to the right by $\begin{bmatrix} 0 & 1 \end{bmatrix}^T$. We obtain

$$A'_{000} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ \ A'_{001} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{43}$$

We see that we cannot reach a valid next state if $\gamma[i] = 0$, so there is an edge between $\{1\}$ at i and $\varnothing$ at $i + 1$ for $\gamma[i] = 0$. If $\gamma[i] = 1$, both states can be reached. Therefore, we draw an edge between $\{1\}$ at i and $\{0, 1\}$ at $i + 1$ for $\gamma[i] = 1$. The other edges of Fig. 3 can be derived in a similar way.

Matrices $B_{00}, B_{01}, B_{10}, B_{11}$ of (41) can be derived from Fig. 3 as

$$B_{00} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \ \ B_{01} = B_{10} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 2 \end{bmatrix}, \ \ B_{11} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \ . \tag{44}$$

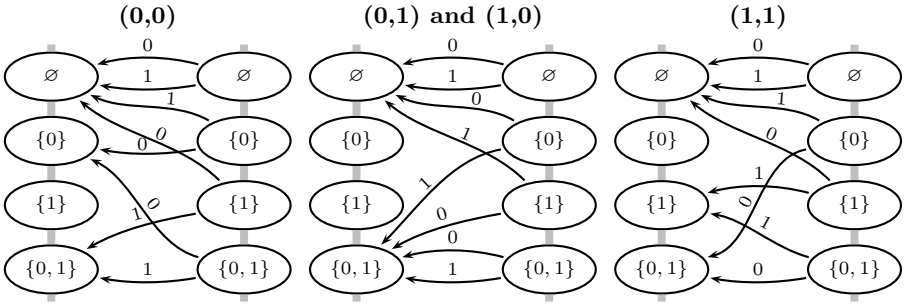

Fig. 3. All possible subgraphs for xdc^+. Vertices correspond to valid sets of states $S[i]$. There is one edge for every output difference bit $\gamma[i]$. Above each subgraph, the value of $(\alpha[i], \beta[i])$ is given in bold.

If the input differences are very sparse or very dense, (41) can be sped up by using the following expressions for the powers of matrices:

$$B_{00}^k = \begin{bmatrix} 1 & k-1 & k \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \; B_{01}^k = B_{10}^k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2^{k-1} & 2^{k-1} & 2^k \end{bmatrix}, \\ B_{11}^k = \begin{bmatrix} 0 & 0 & 0 \\ k-1 & 1 & k \\ 1 & 0 & 1 \end{bmatrix} . \tag{45}$$

This way, we obtain an algorithm with a time complexity that is linear in the number of non-zero input bits. As such, our algorithm always outperforms the naive exponential time algorithm, as well as the exponential time algorithm of [3] that only works for some input differences.

Let $L = [1\ 1]$ and $C = [1\ 0]^T$. We illustrate our method by recalculating the example given in [3]:

$$\mathrm{xdc}^+(\texttt{0x1000010402000000}, \texttt{0x0000000000000000}) \tag{46}$$
$$= L \cdot B_{00}^3 \cdot B_{10} \cdot B_{00}^{19} \cdot B_{10} \cdot B_{00}^5 \cdot B_{10} \cdot B_{00}^8 \cdot B_{10} \cdot B_{00}^{25} \cdot C \tag{47}$$
$$= 5880 \tag{48}$$

5.5 Calculation of $\mathrm{adc}^\oplus$

We can also calculate $\mathrm{adc}^\oplus$, which is the number of output differences for xor, when all differences are expressed using addition modulo 2^n. As the matrices $A_{w[i]}^*$ for $\mathrm{adp}^\oplus$ are of dimension 8×8, the matrices $B_{w[i]}$ of $\mathrm{adc}^\oplus$ would be of dimension $(2^8-1) \times (2^8-1) = 255 \times 255$. However, we find that only 24 out of 255 states are accessible. Furthermore, we find that all 24 accessible states are equivalent to 2 states. In the end, we obtain the following 2×2 matrices:

$$B_{00} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \; B_{01} = B_{10} = B_{11} = \begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix} . \tag{49}$$

These matrices $B_{w[i]}$ are consistent with Theorem 2 of [23]. Although the end result is simple, this example encompasses many of the techniques presented in this paper.

6 Conclusion

In Sect. 2, we introduced the concept of an S-function, for which we build a framework in this paper. In Sect. 3, we analyzed the differential probability xdp^+ of addition modulo 2^n, when differences are expressed using xor. This probability was derived using graph theory, and calculated using matrix multiplications. We showed not only how to derive the matrices in an automated way, but also give an algorithm to minimize their size. The results are consistent with [23]. This technique was extended to an arbitrary number of inputs and to several related constructions, including an entire step of SHA-1. A precursor of the methods in this section was already used for the cryptanalysis of SHA-1 [12,30]. We are unaware of any other fully systematic and efficient framework for the differential cryptanalysis of S-functions using xor differences.

Using the proposed framework, we studied the differential probability $\text{adp}^{\oplus}$ of xor when differences are expressed using addition modulo 2^n in Sect 4. To the best of our knowledge, this paper is the first to obtain this result in a constructive way. We verified that our matrices correspond to those obtained in [23]. As these techniques can easily be generalized, this paper provides the first known systematic treatment of the differential cryptanalysis of S-functions using additive differences.

Finally, in Sect. 5, we showed how the number of output differences with non-zero probability can be calculated. An exponential-in-n algorithm was already used for this problem in the cryptanalysis of Threefish [3]. As far as we know, this paper is the first to present an algorithm for this with a time complexity that is linear in the number of non-zero bits.

Acknowledgments. The authors would like to thank their colleagues at COSIC, and Vincent Rijmen in particular, for the fruitful discussions, as well as the anonymous reviewers for their detailed comments and suggestions. Thanks to James Quah for pointing out an error in one of the matrices of Appendix A, and for several suggestions on how to improve the text.

References

1. R. J. Anderson, editor. *Fast Software Encryption, Cambridge Security Workshop, Cambridge, UK, December 9-11, 1993, Proceedings*, volume 809 of *Lecture Notes in Computer Science*. Springer, 1994.
2. J.-P. Aumasson, C. Calik, W. Meier, O. Ozen, R. C.-W. Phan, and K. Varıcı. Improved Cryptanalysis of Skein. Cryptology ePrint Archive, Report 2009/438, 2009. `http://eprint.iacr.org/`.
3. J.-P. Aumasson, Çagdas Çalik, W. Meier, O. Özen, R. C.-W. Phan, and K. Varıcı. Improved Cryptanalysis of Skein. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 542–559. Springer, 2009.
4. J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. SHA-3 proposal BLAKE. Submission to the NIST SHA-3 Competition (Round 2), 2008.
5. D. J. Bernstein. The Salsa20 Family of Stream Ciphers. In M. J. B. Robshaw and O. Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2008.
6. D. J. Bernstein. CubeHash specification (2.B.1). Submission to the NIST SHA-3 Competition (Round 2), 2009.
7. E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.
8. E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, and M. Videau. Shabal, a Submission to NIST's Cryptographic Hash Algorithm Competition. Submission to the NIST SHA-3 Competition (Round 2), 2008.

9. E. W. Chittenden. On the number of paths in a finite partially ordered set. *The American Mathematical Monthly*, 54(7):404–405, 1947.
10. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
11. M. Daum. Narrow T-Functions. In H. Gilbert and H. Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 50–67. Springer, 2005.
12. C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.
13. N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein Hash Function Family. Submission to the NIST SHA-3 Competition (Round 2), 2009.
14. D. Gligoroski, V. Klima, S. J. Knapskog, M. El-Hadedy, J. Amundsen, and S. F. Mjølsnes. Cryptographic Hash Function BLUE MIDNIGHT WISH. Submission to the NIST SHA-3 Competition (Round 2), 2009.
15. S. Hong, D. Hong, Y. Ko, D. Chang, W. Lee, and S. Lee. Differential Cryptanalysis of TEA and XTEA. In J. I. Lim and D. H. Lee, editors, *ICISC*, volume 2971 of *Lecture Notes in Computer Science*, pages 402–417. Springer, 2003.
16. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley, 2006.
17. D. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3):161 – 190, 1954.
18. S. Indesteege and B. Preneel. Practical Collisions for EnRUPT. *Journal of Cryptology (to appear)*, 23(3), 2010.
19. A. Klimov and A. Shamir. Cryptographic Applications of T-Functions. In M. Matsui and R. J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 2003.
20. G. Leurent, C. Bouillaguet, and P.-A. Fouque. SIMD Is a Message Digest. Submission to the NIST SHA-3 Competition (Round 2), 2009.
21. H. Lipmaa. On Differential Properties of Pseudo-Hadamard Transform and Related Mappings. In A. Menezes and P. Sarkar, editors, *INDOCRYPT*, volume 2551 of *Lecture Notes in Computer Science*, pages 48–61. Springer, 2002.
22. H. Lipmaa and S. Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In M. Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.
23. H. Lipmaa, J. Wallén, and P. Dumas. On the Additive Differential Probability of Exclusive-Or. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2004.
24. R. F. Lyon. Two's Complement Pipeline Multipliers. *IEEE Transactions on Communications*, 24(4):418–425, April 1976.
25. R. F. Lyon. A bit-serial architectural methodology for signal processing. In J. P. Gray, editor, *VLSI-81*, pages 131–140. Academic Press, 1981.
26. M. Matsui and A. Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In *EUROCRYPT*, pages 81–91, 1992.
27. G. H. Mealy. A method for synthesizing sequential circuits. *Bell Systems Technical Journal*, 34:1045–1079, 1955.
28. A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
29. E. F. Moore. Gedanken experiments on sequential machines. *Automata Studies*, pages 129–153, 1956.
30. N. Mouha, C. De Cannière, S. Indesteege, and B. Preneel. Finding Collisions for a 45-Step Simplified HAS-V. In H. Y. Youm and M. Yung, editors, *WISA*, volume 5932 of *Lecture Notes in Computer Science*, pages 206–225. Springer, 2009.
31. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007. Available: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf` (2008/10/17).
32. R. M. Needham and D. J. Wheeler. Tea extensions. Computer Laboratory, Cambridge University, England, 1997. `http://www.movable-type.co.uk/scripts/xtea.pdf`.

33. S. O'Neil, K. Nohl, and L. Henzen. EnRUPT Hash Function Specification. Submission to the NIST SHA-3 Competition (Round 1), 2008.
34. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. *The Twofish encryption algorithm: a 128-bit block cipher.* John Wiley & Sons, Inc., New York, NY, USA, 1999.

A Matrices for xdp$^+$

The four distinct matrices $A_{w[i]}$ obtained for xdp$^+$ in Sect. 3.4 are given in (50). The remaining matrices can be derived using $A_{001} = A_{010} = A_{100}$ and $A_{011} = A_{101} = A_{110}$.

$$A_{000} = \begin{bmatrix} 3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 3 \end{bmatrix}, \quad A_{001} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}, \quad A_{011} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}, \quad A_{111} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{50}$$

Similarly, we give the four distinct matrices $A'_{w[i]}$ of Sect. 3.4 in (51). The remaining matrices satisfy $A'_{001} = A'_{010} = A'_{100}$ and $A'_{011} = A'_{101} = A'_{110}$.

$$A'_{000} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad A'_{001} = \frac{1}{2}\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad A'_{011} = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, \quad A'_{111} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}. \tag{51}$$

B All Possible Subgraphs for xdp$^+$

All possible subgraphs for xdp$^+$ are given in Fig. 4.

C Computation of xdp$^+$ with Multiple Inputs.

In Sect. 3, we showed how to compute the probability $\text{xdp}^+(\alpha, \beta \to \gamma)$, by introducing S-functions and using techniques based on graph theory and matrix multiplications. In the same way, we can also evaluate the probability $\text{xdp}^+(\alpha[i], \beta[i], \zeta[i], \ldots \to \gamma[i])$ for multiple inputs. We illustrate this for the simplest case of three inputs. We follow the same basic steps from Sect. 3 and Sect. 4: construct the S-function, construct the graph and derive the matrices, minimize the matrices, and multiply them to compute the probability.

Let us define

$$S[i] \leftarrow (c_1[i], c_2[i]) \ , \tag{52}$$

$$S[i+1] \leftarrow (c_1[i+1], c_2[i+1]) \ . \tag{53}$$

Then, the S-function corresponding to the case of three inputs x, y, q and output z is:

$$(\Delta^{\oplus} z[i], S[i+1]) = f(x_1[i], y_1[i], q_1[i], \Delta^{\oplus} x[i], \Delta^{\oplus} y[i], \Delta^{\oplus} q[i], S[i]). \quad 0 \le i < n \ . \tag{54}$$

Because we are adding three words in binary, the values for the carries $c_1[i]$ and $c_2[i]$ are both in the set $\{0, 1, 2\}$. The differential $(\alpha[i], \beta[i], \zeta[i] \to \gamma[i])$ at bit position i is written as a bit string $w[i] \leftarrow \alpha[i] \parallel \beta[i] \parallel \zeta[i] \parallel \gamma[i]$. Using this S-function and the corresponding graph, we build the matrices $A_{w[i]}$. After we apply the minimization algorithm (removing inaccessible states and combining equivalent states) we obtain the following minimized matrices:

$$A_{0000} = \begin{bmatrix} 4 & 0 & 0 & 2 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 6 \end{bmatrix}, A_{0001} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}, A_{0010} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}, A_{0011} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 4 & 4 \\ 0 & 0 & 2 & 0 \\ 2 & 0 & 2 & 4 \end{bmatrix},$$

$$A_{0100} = \begin{bmatrix} 0&1&0&0 \\ 0&4&0&0 \\ 0&0&0&0 \\ 0&3&0&0 \end{bmatrix}, A_{0101} = \begin{bmatrix} 2&0&0&0 \\ 4&0&4&4 \\ 0&0&2&0 \\ 2&0&2&4 \end{bmatrix}, A_{0110} = \begin{bmatrix} 2&0&0&0 \\ 4&0&4&4 \\ 0&0&2&0 \\ 2&0&2&4 \end{bmatrix}, A_{0111} = \begin{bmatrix} 0&0&0&0 \\ 0&4&0&0 \\ 0&1&0&0 \\ 0&3&0&0 \end{bmatrix},$$

$$A_{1000} = \begin{bmatrix} 0&1&0&0 \\ 0&4&0&0 \\ 0&0&0&0 \\ 0&3&0&0 \end{bmatrix}, A_{1001} = \begin{bmatrix} 2&0&0&0 \\ 4&0&4&4 \\ 0&0&2&0 \\ 2&0&2&4 \end{bmatrix}, A_{1010} = \begin{bmatrix} 2&0&0&0 \\ 4&0&4&4 \\ 0&0&2&0 \\ 2&0&2&4 \end{bmatrix}, A_{1011} = \begin{bmatrix} 0&0&0&0 \\ 0&4&0&0 \\ 0&1&0&0 \\ 0&3&0&0 \end{bmatrix},$$

$$A_{1100} = \begin{bmatrix} 2&0&0&0 \\ 4&0&4&4 \\ 0&0&2&0 \\ 2&0&2&4 \end{bmatrix}, A_{1101} = \begin{bmatrix} 0&0&0&0 \\ 0&4&0&0 \\ 0&1&0&0 \\ 0&3&0&0 \end{bmatrix}, A_{1110} = \begin{bmatrix} 0&0&0&0 \\ 0&4&0&0 \\ 0&1&0&0 \\ 0&3&0&0 \end{bmatrix}, A_{1111} = \begin{bmatrix} 0&0&0&0 \\ 8&0&0&0 \\ 0&0&4&2 \\ 0&0&4&6 \end{bmatrix}.$$

D Computation of $\mathbf{xdp^{\times 3}}$

Given n-bit words $x_1, \Delta^{\oplus} x$, we can calculate $\Delta^{\oplus} z$ using

$$x_2 \leftarrow x_1 \oplus \Delta^{\oplus} x \ , \tag{55}$$
$$z_1 \leftarrow x_1 \cdot 3 = (x_1 \ll 1) + x_1 \ , \tag{56}$$
$$z_2 \leftarrow x_2 \cdot 3 = (x_2 \ll 1) + x_2 \ , \tag{57}$$
$$\Delta^{\oplus} z \leftarrow z_2 \oplus z_1 \ . \tag{58}$$

We then define $\mathrm{xdp}^{\times 3}(\alpha \to \gamma)$ as

$$\mathrm{xdp}^{\times 3}(\alpha \to \gamma) = \frac{|\{x_1 : \Delta^{\oplus} x = \alpha, \Delta^{\oplus} z = \gamma\}|}{|\{x_1 : \Delta^{\oplus} x = \alpha\}|} \ , \tag{59}$$
$$= 2^{-n} |\{x_1 : \Delta^{\oplus} x = \alpha, \Delta^{\oplus} z = \gamma\}| \ , \tag{60}$$

as there are 2^n values for the n-bit word x_1.

The left shift by one requires one bit of both $x_1[i]$ and $x_2[i]$ to be stored for the calculation of the next output bit. For this, we will use $d_1[i]$ and $d_2[i]$. In general, shifting to the left by i positions requires the i previous inputs to be stored. Therefore, (55)-(58) correspond to the following bit level expressions:

$$x_2[i] \leftarrow x_1[i] \oplus \Delta^{\oplus} x[i] \ , \tag{61}$$
$$z_1[i] \leftarrow x_1[i] \oplus d_1[i] \oplus c_1[i] \ , \tag{62}$$
$$c_1[i+1] \leftarrow (x_1[i] + d_1[i] + c_1[i]) \gg 1 \ , \tag{63}$$
$$d_1[i+1] \leftarrow x_1[i] \ , \tag{64}$$
$$z_2[i] \leftarrow x_2[i] \oplus d_2[i] \oplus c_2[i] \ , \tag{65}$$
$$c_2[i+1] \leftarrow (x_2[i] + d_2[i] + c_2[i]) \gg 1 \ , \tag{66}$$
$$d_2[i+1] \leftarrow x_2[i] \ , \tag{67}$$
$$\Delta^{\oplus} z[i] \leftarrow z_2[i] \oplus z_1[i] \ , \tag{68}$$

where $c_1[0] = c_2[0] = d_1[0] = d_2[0] = 0$. Let us define

$$S[i] \leftarrow (c_1[i], c_2[i], d_1[i], d_2[i]) \ , \tag{69}$$
$$S[i+1] \leftarrow (c_1[i+1], c_2[i+1], d_1[i+1], d_2[i+1]) \ . \tag{70}$$

Then (61)-(68) correspond to the S-function

$$(\Delta^{\oplus} z[i], S[i+1]) = f(x_1[i], \Delta^{\oplus} x[i], S[i]), \qquad 0 \le i < n \ . \tag{71}$$

Each of $c_1[i]$, $c_2[i]$, $d_1[i]$, $d_2[i]$ can be either 0 or 1. After minimizing the 16 states $S[i]$, we obtain only 4 indistinguishable states. Define again 1×4 matrix $L = [1\ 1\ 1\ 1]$ and 4×1 matrix $C = [1\ 0\ 0\ 0]^T$. The differential $(\alpha[i] \to \gamma[i])$ at bit position i is written as a bit string $w[i] \leftarrow \alpha[i] \parallel \gamma[i]$. Then $\mathrm{xdp}^{\times 3}$ is equal to

$$\mathrm{xdp}^{\times 3}(\alpha \to \gamma) = L A^*_{w[n-1]} \cdots A^*_{w[1]} A^*_{w[0]} C \ , \tag{72}$$

where

$$A^*_{00} = \frac{1}{2}\begin{bmatrix} 1&0&2&0\\ 0&0&0&2\\ 1&0&0&0\\ 0&0&0&0 \end{bmatrix}, A^*_{01} = \frac{1}{2}\begin{bmatrix} 0&1&0&0\\ 0&1&0&0\\ 0&0&0&0\\ 0&0&0&0 \end{bmatrix}, A^*_{10} = \frac{1}{2}\begin{bmatrix} 0&0&0&0\\ 0&1&0&0\\ 0&0&0&0\\ 0&1&0&0 \end{bmatrix}, A^*_{11} = \frac{1}{2}\begin{bmatrix} 0&0&0&0\\ 2&0&0&0\\ 0&0&0&1\\ 0&0&2&1 \end{bmatrix} \ .$$

We now illustrate this calculation by example. Let $\alpha =$ `0x12492489` and $\gamma =$ `0x3AEBAEAB`. Then $\mathrm{xdp}^{\times 3}(\alpha \to \gamma) = 2^{-15}$, whereas $\mathrm{xdp}^{+}(\alpha, \alpha \ll 1 \to \gamma) = 2^{-25}$. From this example, we see that approximating the probability calculation of multiplication by a constant using xdp^+, can give a result that is completely different from the actual probability. This motivates the need for the technique that we present in this section. We note there is no loss in generality when we analyze $\mathrm{xdp}^{\times 3}$: the same technique can be automatically applied for $\mathrm{xdp}^{\times \mathrm{C}}$, where C is an arbitrary constant.

E Matrices for $\mathrm{adp}^{\oplus}$

All matrices $A_{w[i]}$ of Sect. 4.4 are given below.

$$A_{000} = \begin{bmatrix} 4&0&0&1&0&1&1&0\\ 0&0&0&1&0&1&0&0\\ 0&0&0&1&0&0&1&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&1&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0 \end{bmatrix}, \quad A_{001} = \begin{bmatrix} 0&1&1&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&1&1&0&4&0&0&1\\ 0&1&0&0&0&0&0&1\\ 0&0&1&0&0&0&0&1\\ 0&0&0&0&0&0&0&1 \end{bmatrix}, \quad A_{010} = \begin{bmatrix} 0&1&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&1&4&0&1&0&0&1\\ 0&1&0&0&0&0&0&1\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&1\\ 0&0&0&0&0&0&0&1 \end{bmatrix},$$

$$A_{011} = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 1&0&0&1&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 1&0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0&0\\ 1&0&0&1&0&1&4&0\\ 0&0&0&1&0&1&0&0 \end{bmatrix}, \quad A_{100} = \begin{bmatrix} 0&0&1&0&1&0&0&0\\ 0&4&1&0&1&0&0&1\\ 0&0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0&1\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&1&0&0&1\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix}, \quad A_{101} = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 1&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 1&0&0&0&0&0&1&0\\ 1&0&0&1&0&4&1&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&1&0&0&1&0 \end{bmatrix},$$

$$A_{110} = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 1&0&0&0&0&1&0&0\\ 1&0&0&0&0&0&1&0\\ 1&0&0&4&0&1&1&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&1&1&0 \end{bmatrix}, \quad A_{111} = \begin{bmatrix} 0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&1&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&1&0&0&1&0&0&0\\ 0&0&1&0&1&0&0&0\\ 0&1&1&0&1&0&0&4 \end{bmatrix}.$$

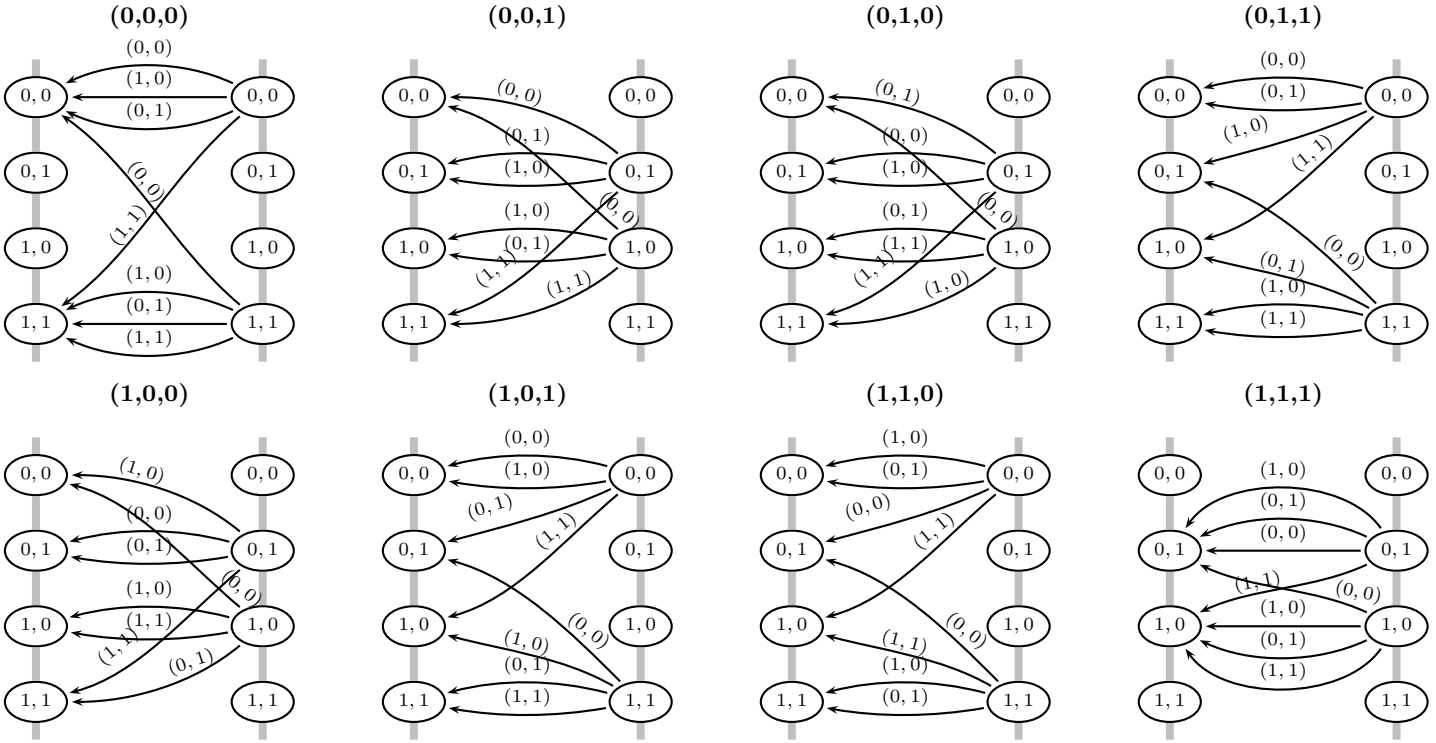

Fig. 4. All possible subgraphs for xdp$^+$. Vertices $(c_1[i], c_2[i])$ correspond to states $S[i]$. There is one edge for every input pair (x_1, y_1). Above each subgraph, the value of $(\alpha[i], \beta[i], \gamma[i])$ is given in bold.